Collaborative Artificial Neural Networks
SIMI-CANN

FACULTY OF
COMPUTER SCIENCE

Cristian Simionescu

Supervisor: Associate Prof. Adrian Iftene

Faculty of Computer Science
Alexandru Ioan Cuza University

This thesis is submitted for the degree of Bachelor of Science

Faculty of Computer Science July 2018
This thesis is dedicated to my parents whose good examples have always taught me only hard work will bring you closer to the things you aspire to achieve. I also dedicate this to Madalina who has always been a constant source of support and encouragement during the challenges of my whole college life.
Acknowledgements

I would like to sincerely thank my thesis adviser, Dr. Adrian Iftene, for his guidance and support throughout this study and especially for the trust he put in me. To friends and colleagues, thank you for your encouragement in the many difficult moments. I also want to thank Cristina Jari for her help.

This thesis is only the beginning of my journey.
Abstract

Taking inspiration from the brain and to some extend human nature, this experiment seeks to reproduce the noticeable improvement tasks experience when performed collaboratively, in the context of working with Artificial Neural Networks.

Parts of the brain, when stimulated by different stimuli, work together to reach better decisions and understanding. Groups of individuals having different view points and knowledge can collectively reach better conclusions by virtue of debating and deliberating. These are the processes I have attempted to emulate in my experiments by trying multiple collaboration techniques, networks and problems to find out if multiple agents combined produce noticeably improved results compared to them working separately.

The name I coined for systems which combine the results of multiple agents by means of Artificial Neural Networks is the name shared by this thesis: "Collaborative Artificial Neural Networks" (CANN).

The methods I came up with and will be presented in this paper are the following three:

1. Simple Collaboration Network (SCN), as the name suggests this is a very simple approach to modeling the system I envisioned;

2. Empathic Collaboration Network (ECN), an improved version of SCN which requires a larger model and more training but obtains better results;

3. Progressive Collaboration Network (PCN), a slightly more complex and ambitious method with which I reached some interesting but mixed initial results.

The problems chosen to demonstrate the proof-of-concept were image classification using the ImageNet dataset as well as economic metrics prediction using historic US data.

As far as I was able to research, the ideas and concepts presented here haven’t been tried before or there aren’t any easily found publications about them. As such I consider that the ideas presented here, experiments and their results posses a considerable amount of novelty.

**Key words**— Collaborative Artificial Neural Networks, Feedforward Network, SCN, ECN, PCN, ImageNet
Contribution

In terms of original contributions, the thesis contains multiple instances of my personal ideas and work. Starting from the Collaborative Artificial Neural Networks paradigm and general concept of connecting multiple pre-trained intelligent agents using a neural network to improve their results.

Stemming from this, the three methods of creating such a system that I have came up with so far are also my own original contribution, starting from idea to proof-of-concept implementation:

1. Simple Collaboration Network;
2. Empathic Collaboration Network;
3. Progressive Collaboration Network;

In order to show the improvements this approach brings to the field of Artificial Intelligence, I included in the research project implementations that tackle the problem of object recognition using the 2013 ImageNet dataset consisting of over 160,000 annotated images.

From the very first experiment, the results were above the baseline achieved by well known established models which are considered to be reference points in the field, proving that using CANN can outperform individual models. The same effect has been obtained when attempted to handle two different prediction problems at the same time using historic US economic data and Zillow’s real-estate dataset in order to predict future economic metrics and real-estate prices. Since a lot less work towards solving these problems can be found, I chose to create my own agents to use as components in the CANN system.

I have also implemented various auxiliary tools used for data preprocessing and result visualization.

As seen in the results detailed in this thesis, the paradigm presents itself as being a valid technique to be used in order to significantly improve performance of seemingly any narrow-AI if we have access to multiple agents and posses the expertise required to design and implement a Collaborative Artificial Neural Network compatible with our data and composing models.
## Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols</td>
<td>xiii</td>
</tr>
<tr>
<td>1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>2 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Related Work</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Novelty</td>
<td>7</td>
</tr>
<tr>
<td>2.3 General purpose</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Solution summary</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Outline</td>
<td>8</td>
</tr>
<tr>
<td>3 Simple Collaboration Network</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Theory</td>
<td>10</td>
</tr>
<tr>
<td>3.3 Implementation</td>
<td>13</td>
</tr>
<tr>
<td>3.3.1 ImageNet</td>
<td>14</td>
</tr>
<tr>
<td>3.3.2 Zillow’s &amp; Inflation</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Conclusions</td>
<td>35</td>
</tr>
<tr>
<td>4 Empathic Collaboration Network</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Theory</td>
<td>39</td>
</tr>
<tr>
<td>4.3 Implementation</td>
<td>39</td>
</tr>
<tr>
<td>4.3.1 ImageNet</td>
<td>40</td>
</tr>
<tr>
<td>4.3.2 Zillow’s &amp; Inflation</td>
<td>46</td>
</tr>
<tr>
<td>4.4 Conclusions</td>
<td>49</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5 Progressive Collaboration Network</td>
<td>51</td>
</tr>
<tr>
<td>5.1 Description</td>
<td>51</td>
</tr>
<tr>
<td>5.2 Preliminary results</td>
<td>53</td>
</tr>
<tr>
<td>5.3 Preliminary conclusions</td>
<td>54</td>
</tr>
</tbody>
</table>

References
Symbols

Acronyms / Abbreviations

AGI  Artificial General Intelligence
AI   Artificial Intelligence
ANN  Artificial Neural Network
CANN Collaborative Artificial Neural Network
ECN  Empathic Collaboration Network
FNN  Feedforward Neural Network
MAE  Mean Absolute Error
ML   Machine Learning
MSE  Mean Square Error
NN   Neural Network
PCN  Progressive Collaboration Network
SCN  Simple Collaboration Network
Chapter 1

Motivation

The current state of affairs in the domain of Artificial Intelligence (AI) is dominated almost entirely by development and improvement of narrow AI and some small interest in theoretical hypothesizing regarding wide AI also known as Artificial General Intelligence (AGI).

Narrow AI refers to agents generally capable of performing only one task or limited to operating in a small, predefined range. All "intelligent" systems we know and use today can at best be considered narrow AI, we are currently unable of generalizing the intelligence of our agents. While AI might seem to get more proficient and complex, there is no clear indication that our current efforts and research directions lead us on a path where machine intelligence reaches the next level.

While AGI’s, capable of performing any task, are currently limited to the realm of thought experiments and science fiction, I believe this to be just a matter of time, furthermore I think the way to this new age of AI is through improvement and development of more advanced and efficient narrow AI’s, these agents eventually being the building blocks for true Artificial Intelligence.

A big limitation faced by Machine Learning models comes from the input data we have to work with, from insufficient data to noise or a distorted data distribution. In real-life scenarios another problem arises. Let’s take a few simple examples:

1. When trying to analyze pictures of flowers, while a well-made model could classify images of tulips correctly with low error rate but would almost certainly misclassify crocus as also being tulips due to the very similar appearance of the two species, the usual discerning factor being scale and scent. These factors aren’t represented in the input even though they would be necessary for a better decision.
2. Let’s think of another scenario, a narrow AI created to **predict** real estate prices, trained using historical data. While the agent may be able to perform its task decently, in the real world there are numerous **extrinsic** factors which affect these prices such as inflation or birthrate for example. These mechanisms simply aren’t present in the data we work with.

One naive answer to this problem is to also collect data on these, initially unused, factors then **create** and **train** a new network using this input, usually from **scratch**. The process would create an even **bigger** model which generally leads to significantly more difficult training and slower responses from the AI. We would also have to **discard** all previous networks making the allocated **resources** for those projects an objective waste. Another problem with this approach is the fact that the new network would be virtually unusable or very ineffective if, for some cases, we are not able to get a hold of auxiliary data to **aggregate** to our original dataset.

The answer I propose tries to address these cons. Firstly, due to increased interest and research in the field of ANN’s, we theoretically have access to a plethora of narrow AI and intelligent agents designed and built for a **wide array** of specific purposes. All the **methods** I propose strive to dismiss as little as possible from the effort and time put into the creating of these systems, an important component of the way I evaluate any of the methods presented here if the approach allows to use a **pre-trained** network "as is" and with as little **additional training** as possible, trying to create as small of a model as possible.

In addition to everything mentioned above, my motivation for trying these ideas comes from the fact that I believe that we currently posses **computational** power several orders of magnitude above what an AGI would require and we also have the technical and theoretical expertise to create good narrow AI for all situations except the ones where a general intelligence seems appealingly necessary. And since we have reached this apparent **skill-cap** where improvements are generally incremental and small, we should think of **different approaches**. These methods, I think could be used in more complex projects to **increase** the operational range our narrow AI operate in, as well as performance.

So to summarize the general motivation and goals of this experiment:

- My personal **curiosity** of seeing if such an approach would work;
- Improving the results of existing networks without **altering** them, allowing usage of **black box** AI agents and **reusage** of great works in the field of ANN’s;
- Trying to find a way to **integrate** more features in our data or auxiliary networks by means of **extension** and not refactoring;
• A very important goal I have set out for this project is that the required training time is as small as possible compared to the training required by agents with similar performance;

• Any positive result would lose most of its value if compared to the baseline agents, the system would require extensive additional resources or computing time;

• Finding if this idea yields the positive results I am interested in for future more complex projects and publications.
Chapter 2

Introduction

Artificial Neural Networks (ANN) are, as the name suggests, systems that try to model and reproduce the way biological neural networks (NN) are known to function in nature. These consist of neurons, which are nature’s equivalent of processor cores but seemingly also serve as storage and memory units. Each of these neurons produce sequential electrical signals which can be seen as real-valued activations. Some neurons produce their activation in reaction to sensory information coming from outside the brain (such as the senses), others get their input, through axons that have the role of weighing the values, from other active neurons and in turn get activated spreading the signals thought the brain.

Using NN as inspiration, we are able to create AI by employing various different techniques and optimizations giving us Feedforward Neural Networks (FNN), Recurrent Neural Networks, Convolutional Neural Networks and others.

Looking at society and nature we will notice that complex systems tend to be formed of simpler components working together, aware of this cooperation taking place or not. Examples range from the classic way of how ant colonies function to people in companies working together or more interestingly how parts of the brain each specialize for different tasks "collaborating" to produce our thoughts, feelings and senses. Let’s take dancing for example: it is a fairly difficult activity to perform, to dance we have to use our primary motor cortex, the primary sensory cortex and two other regions of the brain called the caudate nucleus and thalamus. Even the temporal lobe has to help with spacial awareness while performing the sequence of movements a dance requires.

This is what inspired the idea of having ANN cooperate to perform more difficult tasks or increase performance of the involved networks while working together compared to when working separately. Idea I have suggestively named "Collaborative Artificial Neural Network"
This thesis seeks to attempt a different path not simply improve upon existing work or "reinvent the wheel".

2.1 Related Work

I initially started and finished this thesis without finding any similar work but, with the help of some of my colleagues I managed to discover the works of others in the domain of Committee Machines (Yu Hen Hu & Jenq-Neng Hwang, 2001 [6]). Different types of these systems have been proposed such as Mixture of Experts (Jacobs et al., 1991 [5]) and Neural Network Ensembles (Hansen & Salamon, 1990 [3]).

To briefly describe Committee Machines, they are a method which uses a 'divide and conquer' strategy in which the responses of multiple AI (experts) are combined or used to form a single result. The combined output of the committee machine is supposed to be superior to the one of its components if separated. This paradigm of using multiple intelligent agents to reach better performance englobes my general idea and purpose of CANNs. Committee Machines are generally classified to be of two types:

1. **Static**, where only the output of the involved agents is used, the first method I will describe in the thesis falls under this classification. Progressive Collaboration Networks could also be seen as static to some extent since they don’t work with the input signal. This class of systems usually takes one problem and separates it in subproblems and subsets of input data to be used, then the outputs of different predictors are linearly combined to produce an overall output. While the general idea is similar, the direction and use case of this type of systems is more oriented towards being used on one problem at a time, dividing the problem then linearly combining the subsystem results, while my approach suggests a more general method that can be used for the before mentioned use case, but also others when we want to tackle multiple problems at once or add additional parameters to our system without changing the components.

2. **Dynamic**, this class of Committee Machines will also make use of the input received by its components, using a gating mechanism to combine the responses. The second approach presented in this paper can be considered Dynamic. Previous works used the gate as a means to switch between which of the outputs to be chosen. Training of the composing neural networks is done in this framework, differentiating my approaches by the fact that original agents are pre-trained AI and the output of this agents is "lost" in the non-linear mechanism of CANN to produce the final output.
2.2 Novelty

From all I could gather by means of various sources such as the internet, professors and colleagues I could not find any instances of this particular idea discussed or experimented and therefore the ideas themselves can be considered to have quite a level of novelty. As well as the CANN paradigm which is similar in concept with Mixture of Experts, Neural Network Ensembles and Boosting algorithms but differs in direction and scope.

More precisely, the three approaches of combining two or more ANN without altering the components in order to achieve better performance or increase the range in which those agents can operate are all original ideas I have came up with and experimented (SCN,ECN,PCN). I used these methods to reach better results not by simply finding better meta-parameters or using different combinations of optimizations, number of layers, types of neurons and so on, but by extending the networks from the outside. This method can theoretically be used with any AI agent we develop and hence it could be seen as an improvement to be used on all ANN's past, present and future when the situation permits and/or calls for it.

2.3 General purpose

The purpose of this thesis is to present a new paradigm and three technical suggestions which stem from this paradigm. In the paper I have also presented implementations to act as my proof-of-concept as well as all the results of the mentioned experiments. Finally, the final chapter and the final subsections of each chapter analyze and discuss the implications of the results as well as possible future work.

2.4 Solution summary

To summarize the solution, I used for the purpose of this experiment the well known ImageNet dataset and pre-trained known models such as AlexNet, GoogleNet and others, afterwards applying the three collaborative methods described in this paper trying different meta-parameters or optimizations to reach better results.

When it comes to the actual methods, to shortly describe them here, would be:

**SCN** the simplest of them, uses the output of the collaborative components and tries to reach better solutions based on that;

**ECN** similar to SCN but also takes into account the corresponding inputs of the components;
PCN a completely different approach that sadly sacrifices the "black box" flexibility the other two methods benefited from, even if this one too does not require to alter the actual networks that form the system, the best way to describe this method is a "membrane"-like neural structure between the layers of the components;

### 2.5 Outline

The thesis as a whole contains theoretical formulations and practical implementations but also results, conclusions and descriptions of ideas for each of the methods described. I have tried to avoid clunking the paper with general notions or explanations of concepts and their theoretical backbone that are not my own, so I have skipped explaining Artificial Neural Networks, Feedforward Neural Networks, Convolutional Neural Networks, Machine Learning, Supervised/Unsupervised/Reinforcement Learning and other concepts, since I felt that these things are not mine to write about, they don’t make the subject of this thesis and would only have a place if I wanted to bloat the text valuing quantity over quality.

This is what follows in the rest of the paper:

- **Chapter 3**: Presents the first method the Simple Collaboration Network, which will explain the general concept, theoretical backing and some examples of implementation and results of this approach as well as a "journal" of my thinking process, arising problems and my solutions to them. I will also be drawing a conclusion and planned future work.

- **Chapter 4**: Deals with Empathic Collaboration Networks, same as before it will present the idea, theory, implementation and results. Again, I will also be presenting my conclusion and possible future work.

- **Chapter 5**: The last main chapter offers information regarding the most daring of the three approaches the Progressive Collaboration Network, only the general concept and preliminary results will be presented.

- **Final Remarks**: This is the final part of this thesis and the "scoreboard" for the experiments presented before it. I discuss the overall implications of the results and how I envision these ideas going forward.
Chapter 3

Simple Collaboration Network

3.1 Introduction

The idea behind SCN is simple, hence the name, use the decisions reached by multiple agents to improve or "rethink" the outputs given. Reusing an example given before, imagine a AI that predicts future apartment prices in Iasi, now let’s think we also build or have access to a second AI used to estimate future inflation rates in Romania.

While the two would most likely use completely different data and try to extrapolate seemingly unrelated outputs, it is easy to realize that the two objectives of the agents are indirectly connected and influence each other. Anecdotally, if two specialists would be to discuss, one telling the other that according to his calculations the inflation rate in Romania will reach 7.4% by the end of the year and the other would share his results that the average cost of an apartment will rise up to $1,491.73/\text{m}^2$, these experts could probably improve their findings taking into account the other person’s insight, theoretically one could redo this process of reevaluation until a state of equilibrium is reached regarding the results.

In the context of neural networks, this idea translated into the following: having $n$ number of networks each network being composed of:

- $I_i, 1 \geq i \geq n$ the input layer of the network;
- $N_i, 1 \geq i \geq n$ the hidden layers of said network;
- $O_i, 1 \geq i \geq n$ the output layer.

Then the SCN method works as follows, we shall construct a new ANN having $O_1, O_2 \ldots O_n$ act as the input layer of this SCN, the output layer being the same format as the input, I shall
write them as $O'_1, O'_2 \ldots O'_n$ and they will represent the new, hopefully, improved outputs. See diagram 5.1 bellow:

![Diagram of Simple Collaboration Network]

**Fig. 3.1 General idea of how SCN are constructed**

The Simple Collaboration Network itself is a **placeholder** for any kind of Feedforward Neural Network with arbitrary number of layers, neurons, optimization techniques, activation functions or error function. Finding the most optimal combination of these meta-parameters will depend on the problem at hand and the expertise possessed by the creator of the solution, thing that don’t make the topic of this thesis, further work and experimentation is required to develop these systems. This paper will only present these **approaches** and implementations to explain and exemplify the idea as well as provide a **proof-of-concept**.

### 3.2 Theory

The general idea is motivated by the concept that, while we may be unable to find solutions for complex problems, if we are able to solve easier related tasks we might use these subproblems to solve the harder one. These methods work on the fact that output of the system is formed using all the **subsystems** predictions.

$$x_{SCN} = [\hat{y}_1, \hat{y}_2 \ldots \hat{y}_n], \text{ where } \hat{y}_i \text{ is the output of agent } i \text{ and } x_{SCN} \text{ is the input of SCN} \quad (3.1)$$

I want to consider the approaches to be a **meta-algorithm** and as such parts of it can easily be adapted or changed to use different types of NN, cost functions, optimizations or activation functions but, for the purpose of exemplification I have chosen specific ones to formally display.
Since neural networks are often thought of as function estimators, if we would choose our SCN to be a simple Single-Layer Network (it can easily be extended to a deep neural network) we could describe our system as follows:

\[ f_i(x_i) = [y_{i1}, y_{i2} \ldots y_{im}] \]  
\[ z_j = \sum_i w_{ji} \cdot y_{ji}, \text{ net input for neuron } j, \]  
\[ f_{SCN}(f_1(x_1), f_2(x_2) \ldots f_n(x_n)) = [\sigma(z_1), \sigma(z_2) \ldots \sigma(z_m)] \]

Where \( m \) is the number of output variables, \( f_i \) is agent \( i \)'s function approximation and \( x_i \) the input of said agent, \( i = 1 \ldots n \) of and using \( \sigma \) as our activation function.

For the purpose of this experiment I shall only use Feedforward Neural Networks for our CANNs as such let us look further, to the most important part of every Machine Learning model, training, in this case I will be using Backpropagation. This is how training would occur with Mean Squared Error (MSE) as our cost function:

- Compute the cost function using the error from the last layer noted \( L \), I will also denote output variable \( j \) of agent \( i \):
  \[ \delta_{ij}^L = y_{ij}^L \cdot (1 - y_{ij}^L) \cdot (y_{ij}^L - t_{ij}) \]  

- Backpropagate the error to the previous layer \( l \) and compute the gradient for the weights and biases repeat until we reach the first layer:
  \[ \delta_{ij}^l = y_{ij}^l (1 - y_{ij}^l) \cdot \sum_k \delta_{ij}^{l+1} \cdot w_{ij,k}^{l+1} \]  
  \[ \frac{\partial C}{\partial w_{ij,k}} = \delta_{ij}^l \cdot y_{ij}^{l-1} \]  
  \[ \frac{\partial C}{\partial b_{ij}} = \delta_{ij}^l \]  

- The rest of the algorithm continues as known, we adjust the weights and biases using the gradient and an arbitrarily chosen learning rate (\( \eta \)):
  \[ w = w - \eta \cdot \frac{\partial C}{\partial w}, b = b - \eta \cdot \frac{\partial C}{\partial b} \]
We can now see that our Simple Collaboration Network seems to act as a simple extension which aggregates the outputs of the other agents and applies a series of nonlinear modifiers to existing results.

Let’s see why such an approach should produce better outputs. This takes us to the bias–variance trade-off discussion.

![Graphical Illustration of Bias and Variance](image)

Fig. 3.2 Graphical illustration of bias and variance

Let us consider that we can make abstraction or simply ignore the bias-variance status of the composing NNs in order to just analyze if the technique increased or decreased these properties. There are three main scenarios to consider based on the types of agents that help compose the CANN.
3.3 Implementation

1. If we are to use this technique on a set of agents all used to perform the same task, and they perform similarly on the majority of cases, we shouldn’t see a noticeable change in either bias or variance since for this method to produce different results to the ones the agents achieve individually, the original outputs have to differ, otherwise our trained SCN will not modify anything and behave similar to this uninteresting linear function \( f(x) \approx x \).

2. When used agents with same task type, but encounter different outputs, when applying SCN we should expect the same variation in data but a lower bias, this is due to the fact that cases where both models respond correctly or incorrectly, the output will remain the same, but the cases where only one of the outputs is correct it will cause predictions of the other network to reach a result closer to the target, hence reducing bias. Variance should remain unaffected since there is no mechanism that directly modifies this parameter.

3. The last scenario I could find is the case where we use SCN with agents design to approach different problems. This would be the case when trying to expand the context to increase accuracy or range of inputs the agents can be used for. The bias-variance trade-off should again see bias trend to decrease because the number of variables that take part in the computation of the output has increased, but while we would expect variation to increase this won’t be the case since the trained SCN only works with the outputs and because of that there is no apparent reason variation should increase.

In my opinion this is the reason CANN techniques should generally outperform single networks, because it reaches a lower bias-variance balance.

3.3 Implementation

When choosing how to implement all three of the CANN approach in order to test out my ideas, I tried to find the easiest to work with and most documented networks in order to confirm the proof of concept. The datasets I have used are the ImageNet dataset to test multiple agents combined to solve the same problem and two datasets of historical data the "Zillow’s Home Value Prediction" combined with a Timeseries of Interest vs Unemployment & Inflation in the US.
3.3.1 ImageNet

According to Kaggle’s ImageNet competition\(^1\) description, it's estimated that 1.2 trillion photos were taken in 2017. Even if someone took only one second each photo to organize, tag and annotate, it would still take over 38,000 years to classify them all! The ImageNet competition was founded in 2010 with the aim of enhancing the state of the art in object detection and classification by collecting and offering more data to be used in research and development. ImageNet’s massive database of human-annotated images provide researchers the opportunity to compare progress in automatic \textit{image detection} across a wider variety of objects and measures the progress of computer vision for large-scale image detection and classification.

In my implementation of the SCN I have used the following \textbf{technology stack}:

- \textbf{Docker}\(^2\), used to more reliably keep entire development and testing \textit{environments} for my agents since I used models created in various formats and frameworks I had to be able to easily configure my machines for each experiment and change in the used software.

- \textbf{Python} \(^3\) was the goto programming language used through the project, I chose this language since the vast majority of Machine Learning and Scientific Computing frameworks and packages are implemented or made available to this programming language.

- \textbf{Numpy}\(^3\) is the fundamental package for scientific computing with Python, extensively used for processing the input and output data as well as other auxiliary computations.

- \textbf{Caffe}\(^4\) is a deep learning framework that I had to include in order to load and use the pre-trained models such as:

  - Network in Network (NiN\(^5\)) Imagenet Model, which is an implementation of the structure described in "Network In Network" (Min Lin, Qiang Chen, Shuicheng Yan, 2013 \([4]\)).

3.3 Implementation

- **AlexNet** 6 Model another implementation for image classification described in "ImageNet Classification with Deep Convolutional Neural Networks" (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012 [1]).

- **GoogleNet**7 trained neural network which was created following "Going Deeper with Convolutions" (Christian Szegedy et al. 2014 [2]).

  - **Tensorflow**8 similarly to Caffe is an extensive Machine Learning framework, because of the large amount of granular control on how a model is created. I chose to use this package for creating simple implementations of the SCN described above.

  - Auxiliary packages such as os, time, PIL, etc. which are not important to the scope of this thesis.

An important factor to be mentioned is that a substantial amount of consideration has been put into making the Docker environment able to access the GPU in order to allow Caffe, Tensorflow and Numpy to effectuate computations on the machine’s video card, resulting in a considerable increase in training and **forward-passing** speeds.

**Data**

For the purpose of this thesis I chose to use only a small subset of the ImageNet dataset particularly ImageNet’s 2013 Large Scale Visual Recognition Competition (LSVRC) training data, which in itself is a big 1GB, over 160,000 pictures dataset. Using the data offered online at Kaggle.com9, I **preprocessed** the data to save both space and improve the ease of usage of the images. The steps I took are the following:

1. Parse the XML files describing and labeling each image, then according to that information I **cropped** each image into the sub-boxes described for each object labeled in the picture 3.3.

2. Use every pre-trained model I chose to use for this experiment in order to produce their predictions for all the images in my dataset. I did this because I wanted to be able to **parallelize** the process on the two machines I had access to at the time of the experiment. Another benefit of producing the composing agents **predictions** in advance was that I was able to tweak and try different models for my SCN without

---

6https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet
7https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet
8https://www.tensorflow.org/
9https://www.kaggle.com/
having to recompute the outputs of my **components** every time. I saved the resulting matrices locally to be used from that point on.

3. Filtering all the inputs that were not compatible with my existing agents, namely images containing objects and **labels** that the networks I used weren’t trained for, classes that weren’t part of my chosen 1,000 labels. I did this simply to free disk storage space and reduce pointless processing time, this step removed about 50,000 images.

4. The last step had me resolving various problems my technology stack had. Three of my models were **developed** using the Caffe Machine Learning framework and given that I had to adjust the **data** accordingly, more exactly:

   - In the training stage, Caffe’s C++ implementation uses `imread` in OpenCV to read images, which by default reads the color channels in order of B G R. However, in the Python interface, Caffe uses `skimage` to read image, which by default reads the color channels in order of R G B. For this I had to reshape the last three dimensions of the matrix.

   - The before mentioned `skimage` used by Caffe had another known problem of rotating or changing the orientation of the original image. To fix this issue I set the orientation using the orientation of the image **meta-data** wherever it was present.

**Agents**

As I briefly mentioned above, the composing agents that I used were: **NiN**, **AlexNet**, **GoogleNet** as well as the CANN used to combine them. Moving forward, let’s have a more detailed look at the general structure and **components** taking part in this **system**:

1. **AlexNet** is a replication of the model described in the AlexNet publication. This model obtains a top-1 accuracy **57.1%** and a top-5 accuracy **80.2%** on the ImageNet competition validation set. The agent itself is a fairly big one having the model size of 230MB, despite this, on my laptop’s Intel 7700k CPU, the forward-pass time averaged at about **0.673291** seconds, which is a decent speed for AI, made to tackle the ImageNet dataset.

2. **NiN** model is a 4 layer Network in Network model trained on ImageNet dataset, thanks to the replacement of the fully connected layer with a global average pooling layer,
3.3 Implementation

Fig. 3.3 An example image from the ImageNet dataset

this model has greatly reduced parameters, which results in a model of size 29MB, considerably smaller than AlexNet which is 230MB, on my Intel 7700k CPU, the forward-pass time averaged at about 0.42391 seconds which is also an improvement over AlexNet. Despite the small model and quick predictions, the model achieved the top 1 performance on the 2010 validation set of 59.36%, which is again slightly better than AlexNet. According to the creators of this particular trained network the training time of the model is between 4-5 days on a GTX Titan, greatly reduced compared to AlexNet which is known to take up to 10 days of training.

3. GoogleNet model is an implementation of the network described in the GoogleNet publication. This trained model obtains a top-1 accuracy 68.7%, significantly better than the previous two, the consequences of obtaining such results come from the fact that on my CPU the average prediction time is 1.46175 seconds despite having a model size of just 53.5MB.

4. SCN, the collaborative artificial neural network I implemented and trained to demonstrate the proof-of-concept, while I tried various configurations and networks, one of the most stable one was a network with one hidden layer having 256 units. Having
such a small model resulted in the SCN forward-pass taking only 0.005123 seconds on average when using my CPU.

The general structure of the implemented system closely follows the SCN description presented in the introduction of this chapter.

Keep in mind that the accuracy percentages presented above were obtained on the 2010 and 2012 ImageNet competition datasets while I used images from 2013. I chose to use a different dataset for both training and testing in order to better show the improvement the SCN method brings to agents since all components will individually under-perform compared to their initial classification task. When recalculating training and test accuracy for the agents using my own dataset, I reached the following results:

<table>
<thead>
<tr>
<th>Agent</th>
<th>Top 1 accuracy</th>
<th>Top 5 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
<td>test</td>
</tr>
<tr>
<td>AlexNet</td>
<td>34.15%</td>
<td>33.99%</td>
</tr>
<tr>
<td>NiN</td>
<td>33.99%</td>
<td>33.85%</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>42.04%</td>
<td>41.72%</td>
</tr>
</tbody>
</table>

As it can clearly be seen, these results are considerably poorer than the ones mentioned previously, this was to be expected as the training of the agents has been done on a wildly different dataset.

A very important thing to remember moving forward is that the statistics above are the baseline accuracies my method has to beat in order for it to be considered successful.

CANN

Initially, when training my system I took each image, batch, mini-batch or training data and fed it to every composing agents, observing that the exact same computations were effectuated at each training epoch, mainly the forward-passing of the images, I performed these operations beforehand and storing the resulting prediction matrices on disk cut the average epoch time from 3.6816 to 0.1241 seconds.

Afterwards I wanted to check if the Simple Collaboration Network method would actually be able to learn and reach a decent accuracy rate. For this, I first trained a "proof-of-concept" SCN, using a very monotone model and training it with a high learning rate. Its composition was the following:

- The 3,000 input units which are the concatenated outputs of the composing agents.
• A single hidden layer having **1,024 units**.

• Another 3,000 units for the output, again **mirroring** the output’s of the components.

• In this model I used no bias, the first weights layer used the ReLu activation function while the second one didn’t use any activators.

• The cost function used is Cross Entropy with the mention that I applied Softmax to the output first and to reduce loss I preferred to use mini-batch training with a batch-size of 256.

• Additional relevant meta-parameters were:
  
  – learning-rate : **2e-3**.

  – weight initialization was done using the normal distribution with a high **0.25** standard deviation.

  – parameter optimizing was effectuated by Stochastic Gradient Descent.

As it can clearly be seen from the description, the initial model I used was designed entirely to test if the concept was viable. The model is prone to **exploding gradients** and **overshooting** because of its relatively high learning rate.

The performance of this seemingly poorly-designed model was unexpected. Not only was it able to learn, but also to **outperformed** all the composing agents on both training and test accuracy in **8 epochs** of training. The best results of this proof-of-concept system were at epoch **17**:

<table>
<thead>
<tr>
<th>Agent</th>
<th>Top 1 accuracy</th>
<th>Top 5 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
<td>test</td>
</tr>
<tr>
<td>AlexNet</td>
<td>49.39%</td>
<td>49.37%</td>
</tr>
<tr>
<td>NiN</td>
<td>49.21%</td>
<td>49.24%</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>49.41%</td>
<td>49.35%</td>
</tr>
</tbody>
</table>

As we can see, the accuracies seemingly **increased** across the board, the only exception being the Top 5 GoogleNet, but this was entirely expected, as mentioned previously, the power of this technique comes from the fact that it preserves variance but lowers bias.

As expected, the model itself suffers from multiple problems, a thing easily seen when looking at the value of our **cost** plotted to epochs, see Figure 3.4.
Fig. 3.4 Evolution of cost when training the proof-of-concept

The fact that our cost has such a sharp trend upwards only consolidates our assumption that our model trends to make gradients **explode**.

These phenomena can also be seen to occur with a small delay in the actual accuracies:
3.3 Implementation

(a) The Top 1 Train Accuracy for Bad SCN

(b) The Top 1 Test Accuracy for Bad SCN

(c) The Top 5 Train Accuracy for Bad SCN

(d) The Top 5 Test Accuracy for Bad SCN

Fig. 3.5 Bad SCN accuracy metrics plotted against epoch

While the network used in this example should never be used in either the industry or for academic purposes, it did show us some surprising results.

Looking at the differences between training and test accuracy results, we can see that our model manages to generalize very well since the accuracy is similar in both.

We can also notice how Top 5 accuracies seem to all get closer to the highest one, AlexNet and NiN reaching GoogleNet top 5 accuracy when working together compared to when separated. This is the behavior I expected, more precisely I expected the lower performing networks to get closer to whichever model had the highest one from the group.

The unexpected and most interesting result is the fact that while Top 1 accuracies did tend to come to the same values, the value they reached is much higher than any of the individual Top 1 Accuracy which would suggest an interesting case. The case in which an input was incorrectly classified by all agents but the correct class still had high values in the initial output seem to become correctly classified in SCN when the agents collaborate to
Simple Collaboration Network

improve their output.

These observations will become clearer when we take a look at the second network I have tried.

After quickly confirming that the approach is capable of learning, I moved forward to try and create a more well designed model. While here too I have tried to keep it as simple as possible and anyone who wants to implement the SCN method is encouraged to attempt more complex architectures and techniques, I chose to simply take the previous one and improve upon it. Consequently, I will only list what I changed from the previous model to obtain a "Good" SCN example model:

- Reducing the number of hidden units from 1,024 to 512 on the hidden layer.
- The learning-rate was decreased to: 1e-6.
- I also decreased the batch-size to 128.
- Weight initialization was done using the normal distribution with a smaller 0.1 standard deviation compared to the previous 0.25.
- The biggest change in performance was when I changed the Stochastic Gradient Descent optimizer to the Adam parameter optimizer.

This new model did not appear to suffer from exploding gradients, overshooting or other common ANN problems. While this time I was expecting positive results, again SCN surpassed expectations providing great accuracy improvements with a very small additional training time. This time, because of the much lower learning rate and number of hidden units, the model was allowed to be trained for more epochs, this time the best training accuracy was encountered on epoch number 118:

<table>
<thead>
<tr>
<th>Agent</th>
<th>Top 1 accuracy training</th>
<th>Top 1 accuracy test</th>
<th>Top 5 accuracy training</th>
<th>Top 5 accuracy test</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>59.68%</td>
<td>59.82%</td>
<td>73.88%</td>
<td>73.47%</td>
</tr>
<tr>
<td>NiN</td>
<td>59.70%</td>
<td>59.72%</td>
<td>73.72%</td>
<td>73.21%</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>59.78%</td>
<td>59.97%</td>
<td>73.78%</td>
<td>73.52%</td>
</tr>
</tbody>
</table>

Even with this simple architecture, the technique proved again to be powerful enough to produce results. I chose to stop training because the loss started to show a trend upwards. To

Table 3.3 Results of the agents collaborating in a "Good" SCN, Epoch 118
avoid this from happening, the model needs more meta-parameter adjustments, this process however is beyond the scope of the experiment. Another reason for not wanting to use more complex models and longer training is the fact that doing so would defeat the purpose anyone would want to use CANN at all, and that is to use existing, extensively tuned and trained agents. These are usually pretty big in size and offer slow predictions, the scope of using a CANN method is to offer a notable improvement to performance with a minimal computational overhead or development and training time.

Similarly with the "Bad" SCN implementation, we can draw a few clear observations:

1. The model doesn’t seem to either overfit nor underfit the data, having seemingly non-existent generalization error. While this could be a result of the chosen structure for this implementation of the SCN and not due to some inherent CANN, the latter seems far more likely.

2. All three network accuracies seem to trend to the same point, and more importantly the collaborative performance far surpasses the performance of the agents when working separately. This again seems to indicate that using SCN helps treat two cases when it comes to the input data:

   • the cases where the input is correctly classified by at least one of the agents but wrongly by the others. In this case SCN will learn the classification from the networks with correct outputs and this will correct the other networks. This can easily be seen by looking at the fact that all three networks have almost identical accuracies.

   • the cases where the input is incorrectly classified by all networks. Looking at the table above and comparing it to the accuracy obtained when separated we can clearly see the improvement, improvement which is far above the highest scoring network. This difference in accuracy can only be explained by input data which was misclassified by all networks separately but together managed to learn that some combinations of smaller probabilities for a class lead to the correct classification.
Fig. 3.6 Evolution of cost when training the Good SCN implementation

Fig. 3.7 Good SCN accuracy metrics plotted against epoch

(a) The Top 1 Train Accuracy for Good SCN
(b) The Top 1 Test Accuracy for Good SCN
(c) The Top 5 Train Accuracy for Good SCN
(d) The Top 5 Test Accuracy for Good SCN
This network, while very basic, was a tad bit better tuned for the task at hand and this was shown by its whopping $10^{-12}$ accuracy percentile increase across all metrics from the "Bad" SCN implementation. When compared to the baseline performance from Google’s model, which was the best performing one used, if measuring for accuracy, we can see an incredible 69.56% performance increase. This result is only made better by the fact that seemingly the only negative trade-off of using this approach is that the overall CANN will only have to sacrifice an additional 0.005123 seconds of processing time per classification (using my laptop’s CPU, in more practical implementations this value becomes even smaller).

Conclusions

Being the first and also the simplest Collaborative Artificial Neural Network I came up with and implemented, I was expecting for it to either be a flat out failure or for it to hardly reach the baseline metrics after much tuning. This was accentuated by the fact that, for the ImageNet experiment, all the agents used were trained for the same task, of image classification, a scenario I considered to be the most challenging use case for any CANN to obtain positive results.

Looking back at the motivations for building such a system I was pleased to see I managed to obtain what I desired with little to no compromises. More exactly:

- The AI was able to learn;
- I was able to maintain the black-box status of the composing agents, a fact I place much emphasis on since it allows components themselves to be replaced by any kind of intelligent agent and not just Neural Networks as I preferred to use in my experiment.
- Taking some of the best models the field was able to produce for the ImageNet contest, SCN was able to greatly improve their accuracy when having them collaborate.
- The training time of both of my models was very short, for the first one it only took 49 seconds of training on my laptop’s GTX 1070 without any multi-threading optimizations. The improved model taking 4 minutes and 12 seconds of training to reach the results presented previously,
- While this method brings noticeable improvements, it manages to do so without putting much additional strain on working memory requirements or computing time. Both implementations I presented produces a very small sized model of just 7.68MB for the "Good" one and 12.28MB for the "Bad" one. When looking at computing time we see a similar effect, both models averaging 0.008123 seconds per image classification.
Table 3.4 Overview of baseline metrics

<table>
<thead>
<tr>
<th>Agent</th>
<th>Accuracy Top 1</th>
<th>Accuracy Top 5</th>
<th>No. Epoch</th>
<th>Epoch time</th>
<th>Memory</th>
<th>Forward-pass time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>33.99%</td>
<td>53.20%</td>
<td>90</td>
<td>2h24m</td>
<td>230MB</td>
<td>0.673291s</td>
</tr>
<tr>
<td>NiN</td>
<td>33.85%</td>
<td>52.51%</td>
<td>60</td>
<td>1h49m</td>
<td>29MB</td>
<td>0.321491s</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>41.72%</td>
<td>61.79%</td>
<td>250</td>
<td>2h3m</td>
<td>27MB</td>
<td>0.922007s</td>
</tr>
</tbody>
</table>

Table 3.5 Overview of "Bad" SCN metrics

<table>
<thead>
<tr>
<th>Agent</th>
<th>Accuracy Top 1</th>
<th>Accuracy Top 5</th>
<th>Additional training time</th>
<th>Additional resources used</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>49.37%</td>
<td>61.34%</td>
<td>17</td>
<td>12.28MB</td>
</tr>
<tr>
<td>NiN</td>
<td>49.24%</td>
<td>61.20%</td>
<td>17</td>
<td>12.28MB</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>49.35%</td>
<td>61.48%</td>
<td>17</td>
<td>12.28MB</td>
</tr>
</tbody>
</table>

Table 3.6 Overview of "Good" SCN metrics

<table>
<thead>
<tr>
<th>Agent</th>
<th>Accuracy Top 1</th>
<th>Accuracy Top 5</th>
<th>Additional training time</th>
<th>Additional resources used</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>59.82%</td>
<td>73.47%</td>
<td>118</td>
<td>7.68MB</td>
</tr>
<tr>
<td>NiN</td>
<td>59.72%</td>
<td>73.21%</td>
<td>118</td>
<td>7.68MB</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>59.97%</td>
<td>73.52%</td>
<td>118</td>
<td>7.68MB</td>
</tr>
</tbody>
</table>

By looking at the table we can concur that the total final SCN system would have to take up 293.68MB and take 0.928458s for a forward-pass. As we can see, most of the model size is taken up by the AlexNet agent, which we could replace for a smaller, similarly performing agent such as SqueezeNet\textsuperscript{10}, the same goes with the forward-pass time, this value could be enhanced by replacing GoogleNet with a faster classifier. Finding and picking the right components for a CANN system should be a task done while taking into account the requirements and parameters we intend our network to operate within.

I want to mention that I have also experimented with a bigger, more complex network than my SCN. I did not include it in the descriptions and discussions above since its huge size, long training time and slow classification time made it incompatible with the goals I have set out to achieve in this thesis.

\textsuperscript{10}https://arxiv.org/abs/1602.07360
Just for the record the "Big" SCN implementation reached a top 1 accuracy of 65.39% and top 5 accuracy of 80.11% across the board. In my opinion the trade-off we have to compromise for in order to reach these results contradict the purpose of using CANN in the first place. Training this network took 768 epochs that lasted 13.716s each, as well as using up 430MB for the model itself. The forward-pass taking 0.071769s.

In conclusion, it can be seen that Collaborative Artificial Neural Networks and more specifically Simple Collaboration Networks can be a viable method to be used in situations where we have access to or are able to build multiple well performing agents for a task and we want to improve performance of our solution as much as possible by using these components. This offers an alternative method of obtaining better results, other than having to create new very complex and highly tuned models which will not only take a very long time to develop (not even having the assurance we can find better models or optimizations), but which would also take a very long time and use many resources to train.

Further work and research has to be done to find better tuned models and experiment with more complex SCN structures. Considering that witch such small amount of meta-parameter tuning and using a very simple NN had us reach considerable improvements to the baseline it would be interesting to see what this method could maximally reach.

Another research direction that is generally valid for all CANN’s is attempting to construct mega systems, composed with as many agents possible and to see if such a system reaches similar, worse or better results.

### 3.3.2 Zillow’s & Inflation

For this second experiment I wanted to test the effectiveness of Simple Collaboration Networks and Collaborative Artificial Neural Networks in general have when used with agents trained for different tasks, using completely different data and contexts of said data. This experiment proposed an additional challenge in the form of having to find datasets and agents trained on datasets capable of being aggregated together.

Because of this I chose two datasets which I found easy to aggregate without much expertise needed in the field:

1. Federal Reserve Interest Rates, 1954-Present\(^{11}\), (I will call this dataset "Inflation" through the rest of the paper) this is a small dataset taken from Kaggle.com which includes data describing the economic conditions in the United States on a monthly basis since 1954.

\(^{11}\)https://www.kaggle.com/federalreserve/interest-rates/home
2. Zillow’s Zestimate home valuation\textsuperscript{12}. First released 11 years ago, it was created to give consumers as much information as possible about homes and the housing market, marking the first time consumers had access to this type of home value information at no cost. “Zestimates” are estimated home values based on 7.5 million statistical and machine learning models that analyze hundreds of data points on each property.

The logic behind picking these two problems has been touched slightly before in this thesis. By use of anecdotal evidence, we would be led to believe the price of real-estate assets would be in a correlative relationship with economic metrics such as unemployment, inflation, or GDP fluctuation. Having this in mind, I wanted to find out if this phenomenon could be exploited by AI, hence using agents that predict future values should be able to confer additional "contextual intelligence" since one of the predicted values is sufficiently accurate, it would present potentially important information when computing the estimates of the other agent.

The same process can and probably should be recreated using even more agents with assumingly better results since the entire CANN system would have a higher level of intelligence because of the increase in context awareness.

This experiment uses a similar technology stack, therefore I won’t re-explain the parts that are common:

- **Docker**
- **Python3.5**
- **Numpy**
- **Tensorflow**

**Scikit-learn** a simple and efficient set of tools for data mining and data analysis. I included this package in order to create narrow AI models based on a different techniques than neural networks. I did this to show that the composing agents can be replaced by any narrow AI, maintaining the black-box principle. More precisely, I used its tools to analyze, fit and predict using Inflation data.

**XGBoost** a flexible Gradient Boosting python package. XGBoost is used for supervised learning, where we fit the training data to predict a target variable. This package was used to create and train a model for the Zillow problem.

\textsuperscript{12}https://www.kaggle.com/c/zillow-prize-1
3.3 Implementation

Keep in mind that the purpose of this experiment was to test if Collaborative Artificial Neural Network, Simple Collaboration Networks to be exact, can be used to improve results of agents created for entirely different tasks.

Data

This time, the experiment required use of two datasets, again found on Kaggle.com, the Inflation and Zillow’s datasets. Compared to the previous example, the amount of data is far smaller and this is also reflected in the models created for them being much smaller and less complex.

1. Inflation, this is a very small dataset containing interest rates, economic growth, unemployment, and inflation data in the United States from 1954 till march 2017. More exactly, it contains the following features:

   • Goods Domestic Product percentage change from the year before
   • Federal Funds Rate
   • Federal Funds Rate Target, an upper, lower and realistic target set out by the Federal Reserve
   • Unemployment Rate
   • Inflation Rate

Out of these features, when creating my model, I pruned them down to these: Inflation Rate, Unemployment Rate, Federal Funds Rate and GDP.

2. Zillow’s data has been offered on the website for use in a prize competition for data scientist to develop better algorithms for solving this task. The dataset itself contains an impressive set of 100,000 entries each with 57 features. I have used only this training set which contained 80,000 of those entries. The features themselves describe a variety of metrics regarding real estates thought the state of California that have been sold throughout 2016 and 2017.

Building a CANN solution for these tasks proposed a new difficulty in the fact that the data used had to be aggregated correctly, for example feeding the system Inflation data from 1980 together with a transaction from 2016 would not make any sense in regards to the purpose and scope of this experiment. While there may be some correlation still existing between 1980’s economic state and todays prices, the most relevant data, in this case, would have to be from the same year and preferably the same month.
Pre-processing the data was made up of the following steps:

1. Pruning of features which brought little to no improvement towards performance in both datasets.

2. Cleaning the data of entries which had missing or invalid values.

3. Based on Year and Month of entries from both sets, I aggregated the data towards Zillow’s dataset. This meant duplicating and dropping entries from Inflation so that each transaction from Zillow’s had a corresponding entry of economic metrics. The same process had to be applied to both input and output data.

4. Similarly to the ImageNet experiment, I chose to create predictions in advance for all the data using the agents, this saved training time as the each entry only had to be processed individually by its corresponding agent once.

Table 3.7 Example Inflation Entry

<table>
<thead>
<tr>
<th>Year</th>
<th>Month</th>
<th>Effective Federal Funds Rate</th>
<th>GDP</th>
<th>Unemployment Rate</th>
<th>Inflation Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>5</td>
<td>0.37</td>
<td>1.7</td>
<td>4.7</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table 3.8 Example Zillow’s Entry

<table>
<thead>
<tr>
<th>airconditioningtypeid</th>
<th>basementsqft</th>
<th>...</th>
<th>landtaxvaluedollarcnt</th>
<th>taxamount</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>561</td>
<td>...</td>
<td>1245</td>
<td>1245</td>
</tr>
</tbody>
</table>

Agents

For this second experiment the agents used weren’t neural network as I wanted to make clear that the method is supposed to work with any narrow AI you add to it. This was also made possible since the actual data didn’t require as complex a function approximation as neural networks can offer. Let’s have a look at the models used:

1. **XGBoost** is short for “Extreme Gradient Boosting” the model is a set of classification and regression trees using gradient. I used XGBoost to train and predict Zillow’s real-estate sale price. This is a very small model with a mere 411.4 KB in size and with a prediction time of just 0.002412s. For joint dataset the Mean Absolute Error (MAE) of the model when working individually is 0.135459.
2. **Linear Regression**, since the Inception dataset is fairly small I decided to go with a very simple linear approach. The model itself uses economic metrics to predict future Inflation, GDP, Unemployment, etc. This even smaller model uses up 5KB and has a near instant computation time. The Mean Absolute Error of this agent when separated reaches 0.062013.

3. **SCN** the Collaborative Artificial Neural Network used to combine the other two agents. This time I will only present the results of one such implementation. From the various attempts I came to a network which had only one hidden layer with 32 units. The model size is just 3.5KB and it too has a negligible forward-pass time.

Keep in mind that on this experiment the two errors measure different tasks and are not to be compared with each other. The XGBoost MAE represents the error with which the model predicts sale prices while the Linear Regression model predicts economic metrics.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Mean Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.127805</td>
</tr>
<tr>
<td>XGBoost</td>
<td>0.059621</td>
</tr>
</tbody>
</table>

The results the agent’s have before combining them using SCN can already be considered to be good when looking at the error values we see that even the simple XGBoost and Linear Regression models were able to **generalize** while not falling into overfitting.

Similarly to the previous experiment, the values in the table above are to be considered the **baseline** results of my agents.

**CANN**

Since the composing agents and datasets used were much simpler, the training and evaluation process itself became trivial. For this experiment I attempted multiple combinations of neurons/layers, meta parameters, loss functions and so on but I will only present one.

I reached a SCN model capable of **improving** the baseline results in very few iterations and because the improvement was good enough to prove the effectiveness of this CANN technique using multi-task agents, I did not waste additional time fine-tuning or attempting more complex architectures.

As such, the SCN model I used is the following:
• 5 input units composed of 1 value coming from Zillow’s predictor while the remaining 4 belonging to the agent working on Inflation.

• A single layer of 32 hidden units.

• The symmetrical 5 output units corresponding to the improved results

• No bias has been used, the first layer using the sigmoid activation function.

• Since both agents used MAE to measure their results individually, I chose to also use this function for the loss function of the network since minimizing the total MAE of the network would directly correspond to smaller individual error.

• Additional meta-parameters used:
  – learning rate of 1e-5
  – normal distribution weight initialization with a 1e-3 standard deviation
  – Adam Optimizer for parameter updates

As I stated several before, this model is not to be taken as a guide when implementing SCN solutions or any CANN for that matter. A more complex model could have been used or even more tuning can be done if desired, I chose not to since those topics are outside the scope and purpose of this paper.

Despite the fact that it may appear that there are too few parameters outputted by the agents for the network to be able to do any kind of learning, it appears that is not the case. Bellow are the results the SCN method reached in 100 epochs of training, each epoch lasting in average 1.408s:

<table>
<thead>
<tr>
<th>Agent</th>
<th>Mean Absolute Error</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
<td>test</td>
<td></td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.049531</td>
<td>0.050689</td>
<td></td>
</tr>
<tr>
<td>XGBoost</td>
<td>0.025802</td>
<td>0.023122</td>
<td></td>
</tr>
</tbody>
</table>

As it can be seen the error has been more then halved by the use of this approach. While the difference may seem small it is important to remember that this was achieved without having to add additional data, extensive tuning or having to discover new and better models.
so it can theoretically be used with more data and better models and it should **enhance** the results there too.

Similarly with the ImageNet experiment we notice that the model is able to learn and improve the outputs of the composing networks while maintaining the same level of **generalization power**. This would suggest that the anecdotal logic I used to motivate trying this experiment has its roots in real world. It suggests that we were successful in "**widening**" the context range of our agents by combining these two **different** narrow AI we created an AI system with a "wider" behavior.

![Fig. 3.8 Evolution of cost when training the SCN model](image)

The abrupt drops in cost indicate that the data used does not reflect the real distribution or that it contains multiple duplicates. When analyzing this fact I come to the realization that the actual aggregated data had a very strange form. While I did not realize it at first in retrospect the two datasets were not ideally picked because of the fact that Zillow’s dataset contains entries for transactions made from October 2016 till December 2016 and the same interval the following year. This basically means that the Inflation dataset metrics was only used for the information during those months hence we have a very poor variation in data.
Curiously enough as we can see when looking at the evolution of our error our models seemed able to still harness useful information from each other when we had them work **collaboratively**.

Like already mentioned, this extremely small extension to our agents with flawed data to train was still able to reduce the baseline error by more than 50% on both test and training sets. The total size of our system reaches **419.9KB** in size (the vast majority of which belonging to just one of the agents) and a average forward-pass time of **0.003212s** using my CPU.

**Conclusions**

This second Collaborative Artificial Neural Network implementation of the Simple Collaboration Network technique brings both new implications regarding the **applicability** of the method but it also reinforces results from the previous experiment.

What I mean by this is that we reaffirm the advantages of SCN or to an extend CANN’s in general:

- Maintaining the **black-box** propriety of our components, making them replaceable out of the box with any new and better agents as the field advances.

- The method seems to constantly **improve** results of our agents, maintaining the variance and decreasing its bias.

- Again the negative effects of applying this method being **negligible**, mainly the added model size and processing time.
• The resources required to create the system are small, from development time and expertize to training time, data and hardware requirements.

This experiment however also presents a limitation in the fact that we can only apply this method with agents that do have an apparent correlation in the tasks and data they use. Also we have to be able to obtain data which we are able to aggregate in a meaningful way. For example I don’t believe AI created to play Mario and Tetris would benefit from being connected in such a system, the way their input data is usually modeled does not have any relationship and this relationship isn’t visible in the real world either.

3.4 Conclusions

When drawing the line and looking at the results of both experiments we can conclude the fact that Simple Collaboration Networks have a clear set of use cases but also limits. One major benefit seen right away is the fact that while research keep being put into creating better and better AI, this method will only become better since it depends directly to the components it uses.

For exemplification if using current established models that peak at 70% accuracy for the same task, and applying SCN we would reach 80% accuracy, if better performing agents are created that surpass the previous 70% baseline by simply replacing the old ones with these improved models we should expect our overall system to improve past 80% as well. The same principle applies for models that treat different problems/tasks where more interesting research could be conducted in trying to create mega Collaborative Artificial Neural Networks with agents and data from as many tasks and fields as possible, this leads to my curiosity if such an approach could lead us to an apparent generally intelligent system.

Since the expertise level required for developing a working system seems small (fact shown by the fact that improvements were seen with minimal tuning), the only real limitation of applying SCN is our access to aggregatable data and performant models. Happily, with the vast amount of resources and research that is being put into this field we should never have a lack data and agents.
Chapter 4

Empathic Collaboration Network

4.1 Introduction

Empathic Collaboration Networks are similar in concept with the previously presented Simple Collaboration Networks. ECN can be seen as an improved or alternative version of SCN. The same principle of using multiple agents and their decisions to improve their results is at work, the difference lying in the fact that the Empathic variant also makes use of the original input during the process.

I found the naming "Empathic" to be suggestive of the fact that differently to the "Simple" variant, this system also has access to the "reasons" the composing agents reach a certain result. Anecdotally it is meant to resemble the concept of human empathy of understanding what another person is experiencing and where their resulting feelings, ideas or reasonings come from.

While previously SCN would "rethink" the output of an agent using the other agents results, practically using to some sense an obfuscation of that model’s context, here I wanted to experiment if the "awareness" of the system would broaden additionally if also given the original data to work with.

While in the previous segment composing agents had to extrapolate meaning from the outputs in which there may be cases where a correlation existed between input data but through the process of computing the output, the resulting data does not have the same relationship. Empathic networks addresses this by also adding the original input as input for the CANN model so even if the correlation is lost or modified in the outputs, the "width" of the context is maintained.

Using the same anecdote as in the last chapter, if two specialists would be to discuss, one telling the other that according to his calculations the inflation rate in Romania will reach
7.4% by the end of the year and the other would share his results that the average cost of an apartment will rise up to 1,491.73€/m², the economists could probably improve their findings by incorporating the other experts result into his calculations. This is where the SCN anecdote would end, in the case of ECN the two experts would also share the metrics and sources they used to reach their conclusions, metrics which could contain relevant information the other expert did not take into account. Even if all input data is indirectly represented in the calculated output, the other agent might gain better information gain from some of the original features.

When modeling this idea for a neural network I obtained the following, having \( n \) number of networks each network being composed of:

- \( I_i, 1 \geq i \geq n \) the input layer of the network;
- \( N_i, 1 \geq i \geq n \) the hidden layers of said network;
- \( O_i, 1 \geq i \geq n \) the output layer.

The ECN method works as follows, we construct a new ANN having \( O_1, O_2, \ldots, O_n, I_1, I_2, \ldots, I_n \), act as the input layer of this ECN, the output layer being of the form \( O_1, O_2, \ldots, O_n \). I shall write them as \( O'_1, O'_2, \ldots, O'_n \) and will represent the improved outputs. See diagram 5.1 bellow:

![Diagram 4.1 General idea of how ECN are constructed](image)

The Empathic Collaboration Network itself, just like SCN, can be replaced with any kind of Feedforward Neural Network with arbitrary number of layers, neurons, optimization techniques, activation functions or error function if it respects the input and output format presented above. Choosing the best ECN model and fine-tuning it to perform optimally on a given task/tasks are not the subject of this thesis, as such consider the examples and implementations presented to be just **proofs of concept**.
4.2 Theory

The way ECN works and the motivation behind it is very much similar to SCN with some small changes regarding the input of the network.

\[ x_{ECN} = [\hat{y}_1, \hat{y}_2 \ldots \hat{y}_n, x_1, x_2 \ldots x_n], \]  
where \( \hat{x}_i, \hat{y}_i \) represent the input and output of agent \( i \) and \( x_{ECN} \) is the input of ECN \hspace{1cm} (4.1)

Looking at narrow AI’s as function approximations and using a Single-Layer Network as our ECN, our overall system could be described as follows:

\[ f_i(x_i) = [y_{i1}, y_{i2} \ldots y_{im}] \]  \hspace{1cm} (4.2)

\[ z_j = \sum_i w_{ji} \ast y_{ji}, \text{ net input for neuron } j, \]  \hspace{1cm} (4.3)

\[ f_{SCN}(f_1(x_1), f_2(x_2) \ldots f_n(x_n), x_1, x_2 \ldots x_n) = [\sigma(z_1), \sigma(z_2) \ldots \sigma(z_m)] \]  \hspace{1cm} (4.4)

Where \( m \) is the number of output variables, \( f_i \) is agent \( i \)’s function approximation and \( x_i \) the input of said agent, \( i = 1 \ldots n \) of and using \( \sigma \) as our activation function.

The rest of the processes work exactly the same as described in the previous chapter.

Emphatic Collaboration Networks can be seen as an extension to a set of models, aggregating both input and output feeding the formed data to a Neural Network.

This method has the same motivation as SCN for why it behaves the way it does, and that is again the bias-variance discussion. Which from my point of view remains the same. Let’s see why such an approach should produce better outputs. This takes the us to the bias–variance tradeoff discussion once again.

The difference between the two being that because of the increased number of features used, ECN should have increased stability when reacting to new data hence a lower variance. This would mean that while SCN maintained the same level of variance but reduced bias, this method while doing the same initially will sacrifice bias to reduce variance. So this method presents a model that doesn’t lower just one metric but balances the two.

4.3 Implementation

For these experiments I chose to use the same datasets and composing models in order to better compare the methods between them. As such the datasets/problems my ECN
implementation will try to tackle are the ImageNet dataset and "Zillow’s Home Value Prediction" combined with "Timeseries of Interest vs Unemployment & Inflation in the US".

4.3.1 ImageNet

Since I used the same composing agents and datasets, my technology stack remained unchanged:

- Docker
- Python3.5
- Numpy
- Caffe
- Tensorflow
- Auxiliary packages such as os, time, PIL, etc. which are not important to the scope of this thesis.

Data

The same dataset and preprocessing steps have been used.

Agents

I used the same pre-trained agents described in the previous chapter changing the Collaboration Network used:

1. AlexNet
2. NiN
3. GoogleNet
4. ECN, a Collaborative Artificial Neural Network a Feedforward network with one hidden layer having 256 units. Because of the nature of the data, this model’s forward-pass time reached a staggering 1.010651 seconds on average when using my CPU.

The general architecture of the resulting system strictly followed the details presented in the introduction of this chapter.

As a reminder, these are the baseline train and test accuracy results for the agents when used separately:
<table>
<thead>
<tr>
<th>Agent</th>
<th>Top 1 accuracy</th>
<th>Top 5 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
<td>test</td>
</tr>
<tr>
<td>AlexNet</td>
<td>34.15%</td>
<td>33.99%</td>
</tr>
<tr>
<td>NiN</td>
<td>33.99%</td>
<td>33.85%</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>42.04%</td>
<td>41.72%</td>
</tr>
</tbody>
</table>

CANN

Just as before to first test if the model was capable of learning I used a "proof-of-concept" ECN which has a very big gradient descent at the start of training, having high valued weight initialization and a very big learning rate. Sadly the weights explode because of the lack of regularization, the network is as follows:

• \((1,000 + 3 \cdot 255 \cdot 255) \cdot 3\) input units, composed of \(1,000 + 3 \cdot 255 \cdot 255\) (255x255 image with 3 color channels) inputs for each agent times 3 since we are using 3 agents.

• A single hidden layer having 1,024 units.

• 3,000 units for the output, mirroring the output of the components.

• In this model I used no bias, the first weights layer used the ReLu activation function while the second one didn’t use any activators.

• The cost function used is Cross Entropy with the mention that I applied Softmax to the output first, to reduce loss I preferred to use mini-batch training with a batch-size of 256.

• Additional relevant meta-parameters were:
  
  
  – weight initialization was done using the normal distribution with a high 0.25 standard deviation.
  
  – parameter optimizing was effectuated by Stochastic Gradient Descent.

Because of the nature of the problem, I expected ECN to **underperform** heavily since using ECN with agents that perform the same task doesn’t really make sense. If we follow the method description, we can clearly see that we will end up with three copies of each input image that is fed towards the network. Additionally, since the input ECN receives contains
image data, a more complex network structure would be required to effectively process the input, like a Convolutional Neural Network.

The method itself is simply not compatible with the problem, the same can be said about the chosen network structure and the data.

The performance of this doomed to fail implementation came as expected, not only was it not able to learn but it was also much slower and used up a lot more memory for the model. To showcase the abysmal results I stopped training at epoch 17, just like the SCN system:

Table 4.2 Results of the agents when collaborating, Epoch 17

<table>
<thead>
<tr>
<th>Agent</th>
<th>Top 1 accuracy</th>
<th>Top 5 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
<td>test</td>
</tr>
<tr>
<td>AlexNet</td>
<td>4.17%</td>
<td>3.39%</td>
</tr>
<tr>
<td>NiN</td>
<td>4.08%</td>
<td>3.84%</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>7.61%</td>
<td>7.44%</td>
</tr>
</tbody>
</table>

As we can see, the accuracies of the model dropped to a level just slightly better to what a “Random” model would obtain.

The cost values not only started from a very high value but it also exploded to ridiculous levels 4.2.

Fig. 4.2 Evolution of cost when training the proof-of-concept

The fact that our cost has drastic shape suggests that our model simply can’t learn the data given to it.

These phenomena can clearly be seen when looking at the evolution of our accuracy, all graphs showed identical trends and almost identical values, because of this I only included one:
While the network used in this example fails miserably as an ECN, I do believe working solutions could be reached but it would require the system to deviate a little from its original description. More concisely, when approaching problems that have us use agents using the same data, we could change the architecture to only include one copy of the agents input data in the input layer of the ECN since the other copies are redundant and only bloat the model and make it much harder to learn. A second modification this ECN needed is to use a model more fit to work with the data, like a Convolutional network and different optimizers, functions and meta-parameters.

This result was entirely expected and it only exists to show limitations, constrains and reinforce a known idea, that when developing any Machine Learning algorithm we first have to look at the data and model our solution accordingly.

Based on this result we can assume that the second, "Good", ImageNet model from the previous chapter will have a similar fate, as such I will only present the model used and its results, the reasoning and analysis of the performance being the same for both of these models. The "Good" model changed the following:

- Reducing the number of hidden units from 1,024 to 512 on the hidden layer.
- The learning-rate was decreased to: 1e-6.
- I also decreased the batch-size to 128.
- Weight initialization was done using the normal distribution with a smaller 0.1 standard deviation compared to the previous 0.25.
• The biggest change in performance was when I changed the Stochastic Gradient Descent optimizer to the Adam parameter optimizer.

This second model had, as predicted, the same fate as the other ECN implementation, bellow you can see the accuracies obtained in the same epoch, 118, as the "Good" SCN model:

Table 4.3 Results of the agents collaborating in a "Good" ECN, Epoch 118

<table>
<thead>
<tr>
<th>Agent</th>
<th>Top 1 accuracy</th>
<th>Top 5 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
<td>test</td>
</tr>
<tr>
<td>AlexNet</td>
<td>4.68%</td>
<td>4.14%</td>
</tr>
<tr>
<td>NiN</td>
<td>4.88%</td>
<td>4.45%</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>8.41%</td>
<td>8.23%</td>
</tr>
</tbody>
</table>

Accuracy did improve compared to the "Bad" implementation, but by such a small margin that it could have simply been due to a lucky random initialization. The cost and accuracy graphs are very much similar to the ones above, as such I have not included them.

Conclusions

This Emphatic Collaboration Network experiment shouldn’t be seen as a failure by any means. The results obtained simply demonstrated the limitations this method brings. As discussed above the results point towards future work to create a ECN tailored for working with images, more exactly with the ImageNet dataset and related agents.

Looking at the results overview bellow we consolidate the detriments suffered when our model is not compatible with the problems data:

Table 4.4 Overview of baseline metrics

<table>
<thead>
<tr>
<th>Agent</th>
<th>Accuracy</th>
<th>Training time</th>
<th>Resources used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top 1</td>
<td>Top 5</td>
<td>No. Epoch</td>
</tr>
<tr>
<td>AlexNet</td>
<td>33.99%</td>
<td>53.20%</td>
<td>90</td>
</tr>
<tr>
<td>NiN</td>
<td>33.85%</td>
<td>52.51%</td>
<td>60</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>41.72%</td>
<td>61.79%</td>
<td>250</td>
</tr>
</tbody>
</table>

In conclusion we can see that Emphatic Collaboration Networks are not meant to be used in same-task Collaboration Networks. There certainly exists models which would obtain SCN level of result improvements, but this network has to adapt the ECN method
4.3 Implementation

Table 4.5 Overview of "Bad" ECN metrics

<table>
<thead>
<tr>
<th>Agent</th>
<th>Accuracy</th>
<th>Additional training time</th>
<th>Additional resources used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top 1</td>
<td>Top 5</td>
<td>No. Epoch</td>
</tr>
<tr>
<td>AlexNet</td>
<td>3.39%</td>
<td>5.78%</td>
<td>17</td>
</tr>
<tr>
<td>NiN</td>
<td>3.84%</td>
<td>7.45%</td>
<td></td>
</tr>
<tr>
<td>GoogleNet</td>
<td>7.44%</td>
<td>10.12%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6 Overview of "Good" ECN metrics

<table>
<thead>
<tr>
<th>Agent</th>
<th>Accuracy</th>
<th>Additional training time</th>
<th>Additional resources used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top 1</td>
<td>Top 5</td>
<td>No. Epoch</td>
</tr>
<tr>
<td>AlexNet</td>
<td>4.14%</td>
<td>6.17%</td>
<td>118</td>
</tr>
<tr>
<td>NiN</td>
<td>4.45%</td>
<td>7.21%</td>
<td></td>
</tr>
<tr>
<td>GoogleNet</td>
<td>8.23%</td>
<td>11.52%</td>
<td></td>
</tr>
</tbody>
</table>

description to remove the duplicated inputs. Said network must also be viable for learning the given problems data, in this case image data.

From this interesting implications arise, looking at accuracy metrics we can conclude the model suffers from very high bias (because of the high training error). It could be said the model underfits the data. This means that when it comes to Emphatic Networks, a good indicator to preventively know if a network structure could be successfully used is by seeing if the model can learn/fit data used by the composing agents.

This wasn’t a problem for the Simple Collaboration Network since we didn’t work directly with the data. But this does suggest another requirement for both SCN and ECN systems to be effective. The aggregating network has to also be able to fit data similar to the output structure. For example if using SCN for tasks that output images, we could no longer use a simple Feedforward Network as I used in the previous chapter as suggested by the results of this experiment.

This leads the direction of future research into finding out the effectiveness of applying SCN/ECN on different types of data if we construct appropriate models for the task at hand. From evidence seen so far and, as we will see in the next experiment, I am inclined to believe that it is entirely possible to create well performing CANN’s for any task we want.
4.3.2 Zillow’s & Inflation

This second Emphatic Neural Network experiment is meant to showcase the effectiveness this method has when used with agents trained for different tasks, escaping the duplicate input problem from the previous case. This experiment also shows the importance of having a Collaboration model compatible with the data.

This experiment stems directly by the anecdotal example given at the beginning of the chapter. Compared to the ImageNet experiment, we can already see how this time the ECN method seems compatible with the two tasks since they address different tasks and use different data fulfilling the exact motivation behind the idea of the Emphatic Networks.

I wanted to discover if the results of these networks would be clearly better then when used in a SCN system since the composing agents would have more contextual data to base their predictions on.

This experiment uses an already described technology stack, as such I won’t describe it again:

- Docker;
- Python3.5;
- Numpy;
- Tensorflow;
- Scikit-learn;
- XGBoost;

The purpose of this experiment is multi-part. Firstly it has the purpose of showing that ECN’s are a viable method of combining AI agents when using the right network for the right problem. Secondly to find out if this "widening" of the context available to the system produces better results than the smaller Simple Collaboration Network systems.

Data

Just like previously, the same datasets were used

1. Inflation;
2. Zillow’s;
4.3 Implementation

Building a CANN solution combining these tasks was pretty simple only having to concatenate the input data with the agent’s predictions.

Pre-processing followed the same steps as in the last Zillow’s & Inflation experiment and the resulting data having the same structures:

Table 4.7 Example Inflation Entry

<table>
<thead>
<tr>
<th>Year</th>
<th>Month</th>
<th>Effective Federal Funds Rate</th>
<th>GDP</th>
<th>Unemployment Rate</th>
<th>Inflation Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>5</td>
<td>0.37</td>
<td>1.7</td>
<td>4.7</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table 4.8 Example Zillow’s Entry

<table>
<thead>
<tr>
<th>airconditioningtypeid</th>
<th>basementsqft</th>
<th>...</th>
<th>landtaxvaluedollarcnt</th>
<th>taxamount</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>561</td>
<td>...</td>
<td>1245</td>
<td>1245</td>
</tr>
</tbody>
</table>

Agents

The same composing agents have been used, only replacing SCN with ECN:

1. XGBoost;

2. Linear Regression;

3. ECN mostly the same structure with only the described difference of having the inputs of the agents also being part of the input of the CANN, as described in the introduction;

The error values for when the agents work separately remained the same:

Table 4.9 Individual agent error

<table>
<thead>
<tr>
<th>Agent</th>
<th>Mean Absolute Error</th>
<th>training</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>0.127805</td>
<td>0.135459</td>
<td></td>
</tr>
<tr>
<td>XGBoost</td>
<td>0.059621</td>
<td>0.062013</td>
<td></td>
</tr>
</tbody>
</table>

While the results already show that the models produce low error, using SCN we were still able to more then halve this value. I am interested in finding out if ECN would perform even better, as my inspiration implied.
CANN

The structure of the ECN used is this time better fit for training on both the input data and the output data of the agents. The network retained the same structure as before:

- 5 input units composed of 1 value coming from Zillow’s predictor while the remaining 4 belonging to the agent working on Inflation.
- A single layer of 32 hidden units.
- The symmetrical 5 output units corresponding to the improved results
- No bias has been used, the first layer using the sigmoid activation function.
- Since both agents used MAE to measure their results individually, I chose to also use this function for the loss function of the network since minimizing the total MAE of the network would directly correspond to smaller individual error.
- Additional meta-parameters used:
  - learning rate of 1e-5
  - normal distribution weight initialization with a 1e-3 standard deviation
  - Adam Optimizer for parameter updates

Compared to the previous time the model makes use of more input features and variation in input, this would have us believe the model can theoretically better fit the data as it has more information to work with.

This time I stopped training earlier than epoch 100 since the system learned faster and I did not want to overfit. The results bellow were obtained during epoch 34, each epoch taking 1.917s of computation.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Mean Absolute Error</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>training</td>
<td>test</td>
<td></td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.013666</td>
<td>0.016274</td>
<td></td>
</tr>
<tr>
<td>XGBoost</td>
<td>0.019327</td>
<td>0.019814</td>
<td></td>
</tr>
</tbody>
</table>

We can notice that the model was able to fit the data and produce good results. The linear model reduces its error 4 times, like in the previous experiment, the errors were already
very small so the change may not seem so significant. I presume that because of the already low baseline error, XGBoost didn’t have an improvement as good as the Linear Regression model.

To my pleasant surprise, the Emphatic Collaboration Network performed better then the "Simple" variant, just as I predicted. This would suggest that my assumptions regarding the effects of widening the context of the system were correct, meaning that the method itself does work as intended but only when used with the correct components and we create compatible Collaboration Networks.

Conclusions

Using the second Collaborative Artificial Neural Network method, the Emphatic Collaboration Network, we were able to take agents created with completely different purposes in mind, using data different data. This shows that by combining seemingly non-related AI and data we are able to create better, more complex systems that are able to respond to more then one tasks.

This made me wonder of the following idea. If we used enough well performing agents created for various tasks and combined them with a method like ECN or SCN, is there a point from which the resulting system can respond to such a wide range of tasks and with relatively high accuracy that it could pass as an Artificial General Intelligence?

Further work could be done trying to build a large, multi-task Collaboration Network. Just like the previous Zillow’s & Inflation experiment, the results touch upon all the desired objectives described in the motivation.

4.4 Conclusions

In retrospect, the conclusions we are to draw from these two experiments give us a better understanding of both the power of using Emphatic Collaboration Networks but also its limits. This helps us define clearer use-cases.

It also shed some light on additional requirements Simple Collaboration Networks have to fulfill in order for them to work effectively, requirements which weren’t easily noticeable when looking at the results of the two SCN experiments which were to some extend "lucky" coincidences that the picked agents and CANN model were able to work together so well.

Further experiments need to be carried out with networks that were designed to handle the problems data. Since the experiments carried here only stand to prove that this is a required criteria to keep in mind when building CANN networks.
Additional work could be done in seeing if using only a subset of features, from the agent’s input, would perform better than using the entire entry in certain conditions. This idea would represent a form of pruning that could be used for ECN models. (they could also be extended for multi-output agents where we would only be interested in a subset of the output). This would require a redefining for the descriptions of SCN, ECN to become more flexible.
Chapter 5

Progressive Collaboration Network

5.1 Description

Progressive Collaboration Networks are the most "unique" of the Collaborative Artificial Neural Networks methods I have proposed so far. While it still follows the CANN paradigm of using pre-trained agents in order to create a system that operated better together then when separated, the way it goes about combining the agents differs greatly. While the other methods used the agents in their composure as black-boxes they simply took outputs and inputs from, PCN works "inside" the box. PCN can only use agents that are themselves already, neural networks, and it works like this:

1. Imagine we have two Feedforward neural networks ($N_1,N_2$);

2. Create two PCN networks:
   - One with the input shape being the number of neurons in the $N_1$ and output shape of the number of neurons in $N_2$;
   - The second one being the mirror of the previous one, with $N_2$ input neurons and $N_1$ output units;

3. We first do a forward-pass through each network and save the activations of each layer from both components;

4. Take the activation values from the previous step and feed them to the corresponding PCN ($N_1$ activation values to the network which has $N_1$ input neurons);
5. Next we do a second forward-pass thought the networks, this time for each neuron from the network we add a "bias" like input that has the value of the corresponding output unit from its PCN, we shall consider this second output to be the final result; While it is difficult to understand in text the figure below:

![Progressive Collaboration Network](image)

**Fig. 5.1 General idea of how PCN are constructed**

The figure above described the process of creating a PCN when trying to combine two networks $N_1, N_2$ of shape $[1, 1]$ and $[1, 2, 1]$, and the chosen PCN is a Single-Layer Network. All of these specifications can be modified to fit any other set of desired neural networks, I simply chose to exemplify this one for a clearer representation. For reference the red "axons" are bidirectional in this situation.

While the process of creating a PCN system for two neural networks is pretty straightforward, when working with more than two agents we have to start making choices regarding our modeling.

We could simply extend the method above to work with $n$ networks by creating $\binom{n}{2} \times 2$ PCN’s so each component sends and receives inputs from all other components. Another alternative would be to create 1 PCN for each component by having concatenating all activations from the other network into the input of the PCN. I have not yet attempted to create Progressive Collaboration Networks for more than two components but experimenting with these two approaches of creating one is a topic of high interest for me personally when it comes to future work.
As it may or may not be apparent already, the naming of the method is meant to be suggestive of the fact that the network is involved in the "decision making" process at each step. To again use a real life equivalent, this would mirror the process of debating or discussion. Since each participating party contributes with an input of information at each step from the start of the decision process till the final conclusion is reached.

Another interesting aspect that could be analyzed in the future is to not stop the model after its second pass but instead to keep repeating the 4th and 5th step from above until we reach some sort of equilibrium, we could measure this by using a norm to see if the final output changed between steps more then an arbitrarily chosen $\varepsilon$.

5.2 Preliminary results

The reason I have not presented this method in a similar fashion to the previous two is the fact that I was not able to implement a solution using it for the ImageNet and Zillow’s experiment because of the fact that, the pre-trained networks I used were in a "frozen" state, meaning I could not alter or interact with the inner-workings of the networks, fact which was completely acceptable for the other two methods which used the agents in a black-box manner.

As for the second experiment, like I previously mentioned I currently only envision this method to be applicable for neural networks, while I do see combinations of neural network agents and other types of models being viable, using this method with a model such as the Linear Regression one would simply produce a ECN system.

While I was not able to create an implementation for the datasets and agents used in this thesis, I did obtain some preliminary results. Using the exact network described in the figure above (5.1), $N_1$ being trained to multiply any input variable by 2 and $N_2$ created to do the same task (multiply by 2).

These very simple agents would almost perfectly approximate their intended functions in very few iterations and as such I had to use the networks before they reach the correct weight values in order to see if PCN was actually capable of learning and reducing error. Using the networks at 50% of the way to being trained, in this case meaning it was 20 iterations away from perfectly fitting the data, I applied PCN. The resulting system reached perfect fitting in 7 iterations.

Since the original networks were capable of reaching 100% accuracy, we can’t predict if applying the method to agents like the ones used in the ImageNet experiment, would have the same effect as SCN of improving the results past the upper baseline. The results of this
small experiment were positive but due to the nature of the agents and datasets used it would be too soon to say that the method was viable in real scenarios.

5.3 Preliminary conclusions

While the pseudo-experiment can’t be considered proof enough for the viability of the method, we can conclude there is potential in future research regarding this method.

In the case that Progressive Neural Networks are viable, they would possess some interesting characteristics. SCN and ECN are both limited by either the output or input of the datasets they work with and using a large amount of agents which use and produce very different types of data would produce a difficult challenge in the design of the CANN to combine them all, since it would have to be compatible with all the data types used. This limitation should simply not be present in PCN systems as it does not have to either produce the final results or have raw data as input.

PCN has the effect of a dynamic secondary bias for the networks it combines, this making it free of comparability issues with either the data or the actual composing networks.

The immediate downside of using this CANN method over the others is the fact that the black-box propriety of component agents is utterly lost and we now have a restriction on the type of agents that make sense to be used since for non NN models the only generalized way of combining other types of models is by only connecting the input and output to the PCN.

An interesting future research direction in the case Progressive Collaboration Networks prove to be effective, would be experimenting with modifications of the original description. One such modification would be to not blindly connect all neurons with all other neurons, but by a case by case basis choose which parts of a network to connect to the input and which ones to the output of a PCN.
Final remarks

As I described at the very beginning, I set out working on this thesis with a number of predefined goals in mind. While I have also encountered failed experiments which I initially thought would be the best performing ones, such as the Emphatic Collaborative Network method applied to the ImageNet dataset and agents, the negative results arguably gave the most insight regarding in what parameters and to what extend do Collaborative Artificial Neural Networks function as intended.

When looking at the results of SCN and ECN experiments, we can see a general characteristics forming. They both have clear benefits when used with different task agents while using components trained for the same task gave ECN problems, for which I proposed a partial remedy.

A clear limitation that became obvious only after the ImageNet ECN experiment is the fact that the network created to combine the other components has to be compatible and able to learn the type of data it receives. In the case of SCN it only needs to be able to fit the output of the components while with ECN it has the added complexity of also having to be structured in such a way that it is able to process the original input.

Regarding this paper’s goal and if they were achieved:

- "Seeing if such an approach would work", happily, while there have been negative results, the overall conclusion is that Collaborative Networks can indeed work, if used correctly.

- "Improving results of existing great AI agents without altering them", this has too been achieved, with both SCN and ECN we were able to take existing agents and use them in a black-box fashion and produce better results then when the agents were used separately.

- Zillow’s & Inflation experiments showed that we can take a model and additional aggregated data (features) and by means of "Collaboration" we were able to improve its results.
• All well designed models had extremely short training periods, even the intentionally bloated models had relatively short training phases when compared to the composing agents. The exceptions to this being XGBoost and Linear Regression models, mainly due to the way learning differs from neural networks.

• Some interesting implications and ideas have spawned from analyzing the experiments, which will act as inspiration for many future projects, each concluding experiment and/or project has at least presents at least direction to be tried in the future.

The encouraging results are a good starting point for further research work in order to push results like the ImageNet SCN or Zillow’s ECN implementations to the highest it can possible get, and to find different network structures for ECN when used with image data and same task components.

Another direction to be pushed further would be a serious look into PCN’s, since it differs greatly from the other two methods the common findings regarding SCN, ECN do no apply here, as such very little is known of its behavior, use cases and limitations.
References


